VZ-LINK MAGAZINE

NEW ZEALAND

VZ 300/200

COMPUTER USERS CLUB

ISSUE 30

APRIL 89

COMMENT

Well,here we are again with issue 30 of your newsletter.
I hope you enjoy all it has to offer.

Thank you, to all those that supported our last Meeting/Workshop. The next one should be held on Saturday 27 May.But check out the next Issue to confirm this date.

The Software Review below was written before D.S.E stopped supporting the VZ.I am looking into the copyright problems of selling Dick Smith VZ software to club members to raise funds for the club.

I now have the Phone connected and you are welcome to ring me after 6.30 PM if you have problems or just want to chat.
The number is Phone 761-451.

I am still after items for the newsletter,So please have a GO My Thanks to the people who have sent in items,check out the acknowledgements elsewhere in this issue.

GAMES REVIEW

Those of you who have had your VZ's for a long time would know that there was a terrific program available called LEARJET. Unfortunately, for some unknown reason, LEARJET is no longer sold by D.S.E. or for that matter, anyone else.

In LEARJET your wits as a pilot and an adventurer are put to the test when you are challenged to take off and land your LEARJET aircraft. A basic aviation knowledge is assumed, but there is a quick but detailed run down of some of the terms in the comprehensive manual supplied with the tape.

This is a true simulation (by the computer meaning) because the computer at no time gives you a graphics picture of what's going on around you. You rely solely on the instrument panel in front of you which takes up the entire screen. This panel is a mixture of low res. and text and as some instruments couldn't be fitted, a few share the same position. For example, Indicated Air Speed, by pressing certain keys can be changed to show True Air Speed or Mach Speed. The key scan routine is rather cumbersome in that it may take up to five seconds for the computer to register the input. This is for two reasons- 1. the keyboard scan routine is slow; 2. nearly the whole keyboard is used to control the aircraft.

After a high res title page you are presented with a list of ten origin and destination points. The longest journey being from Melbourne to Brisbane at 767 nautical miles and the shortest flight is from Williamtown to Sydney, being 76 nautical miles.

The game is highly involved. A routine flight from Sydney to Melbourne take about an hour to successfully complete. A great program for the retired pilot who wants to fly again without leaving the ground.

The manual is full of facts and figures required to operate the instruments. Runway lengths, fuel calculations etc... are all essential for safe flying.

The program was written by J. Keech and P. Russel and is well worth trying to get hold of.

LOADING TIME: Disk- 27 sec.
Tape-4.52 min.
GRAPHICS.....NA
SOUND.....**
ANIMATION....NA
OVER ALL.....****

MONSTER HUNT

```
1 'MONSTER HUNT
50 CLEAR200
60 GOSUB3950
80 FORZ=1TO150:NEXT
100 CLS:COLOR2,1
110 IFRO=16,QU=1:GOTO2950
120 '
130 GOSUB3060
140 '
150 IFA(RO,5)<>0ORA(RO,6)><0ORA(RO,7)<>0,GOSUB3850
160 IFA(RO,8)=0,210
170 PRINTTAB(3);"LOOK OUT !":PRINT"THERE IS A ";M$(A(RO,8));" HERE
180 IFRND(0)>.8,PRINTM$(A(RO,8));" ATTACKS !":KW=1:GOSUB1670:GOTO80
190 '
200 '
210 IFRND(0)>.94,ST=ST-1:IFST<0,ST=0
220 IFRND(0)>.94,CH=CH-1:IFCH<0,CH=0
230 IFRND(0)>.94,DE=DE-1:IFDE<0,DE=0
240 IFRND(0)>.94,IT=IT-1:IFIT<0,IT=0
250 IFRND(0)>.94,WI=WI-1:IFWI<0,WI=0
260 IFRND(0)>.94,CO=CO-1:IFCO<0,CO=0
270 '
280 PRINT:PRINTTAB(6);"YOUR ATTRIBUTES ARE:-"
290 PRINT"STRENGTH";ST,"CHARISMA";CH
300 PRINT"DEXTERITY";DE,"INTELLECT";IT
310 PRINT"WISDOM";WI,"CONSTITUTION";CO
320 '
330 IFST*CH*DE*IT*WI*CO<>0,380
340 PRINT:PRINT"UNFORTUNATELY, YOU ARE EXHAUSTED
350 PRINT"SO THE ADVENTURE ENDS":QU=2:GOTO2990
360 '
370 'CHECK IF ANYTHING CARRIED
380 FL=0
390 FORJ=1TO5
400 IFP(J)<>0,FL=1
410 NEXTJ
420 IFFL=0,490
430 CA=0
440 PRINT:PRINT"YOU ARE CARRYING:-
450 FORJ=1TO5
460 IFP(J)<>0,PRINTO$(P(J)),:CA=CA+V(P(J))
470 NEXTJ
480 IFCA>0,PRINT"TOTAL VALUE =";:PRINTUSING"$$#####.##";CA
490 PRINT
500 '
510 'PLAYER INPUT
520 KW=0
530 FORZ=1TO150:NEXTZ
540 INPUT"WHAT DO YOU WANT TO DO";A$
550 'ADD SPACES IF < 7 LONG
560 IFA$="",540
570 AS=ASC(A$)
580 M=LEN(A$):IFM<7,A$=A$+" ":GOTO580
590 '
600 '
610 B$=LEFT$(A$,3)
620 IFB$="HEL",PRINT"YOU MUST BE JOKING !":GOTO1050
630 IFB$="QUI",QU=4:GOTO2890
640 '
650 '
660 N=1
670 IFMID$(A$,N,1)<>" ",700
```

```
680 C$=MID$(A$,N+1,3):IFLEFT$(C$,1)<>" ",770
690 IFLEFT$(C$,1)=" ",730
700 IFN<M,N=N+1:GOTO670
710 '
720 'ONE WORD ONLY
730 IFRND(0)>.5,750
740 PRINTA$;" IS NOT ENOUGH":GOTO540
750 PRINT"I NEED MORE THAN ";A$:GOTO540
760 '
770 'NEXT UNDERSTANDS INPUT
780 'KILL OR FIGHT
790 IFB$="KIL"ORB$="FIG",KW=1:GOSUB1670
820 '
830 IFB$="GO "ORB$="MOV"ORB$="RUN",KW=1:GOSUB1080
840 '
850 IFB$="GET"ORB$="TAK"ORB$="LIF",KW=1:GOSUB1200
860 '
870 IFB$="DRO"ORB$="PUT"ORB$="LEA",KW=1:GOSUB1440
880 '
890 '
900 '
910 REM
920 '
930 '
940 '
950 '
960 '
970 '
980 IFKW=1,1050
990 '
1000 '
1010 R=INT(RND(0)*3)
1020 IFR=0,PRINT"IT'S MOT WISE TO ";A$
1030 IFR=1,PRINT"WHY WOULD YOU ";A$
1040 IFR=2,PRINT"I DON'T UNDERSTAND "A$
1050 FORZ=1TO750:NEXTZ
1060 GOTO80
1070 '
1080 ' MOVE
1090 C$=LEFT$(C$,1)
1100 IFC$="N"ANDA(RO,1)=0,PRINT"YOU CAN'T GO NORTH":RETURN
1110 IFC$="S"ANDA(RO,2)=0,PRINT"NO EXIT THAT WAY":RETURN
1120 IFC$="E"ANDA(RO,3)=0,PRINT"NO EXIT TO THE EAST":RETURN
1130 IFC$="W"ANDA(RO,4)=0,PRINT"YOU CAN'T GO THROUGH WALLS":RETURN
1140 IFC$="N",RO=A(RO,1)
1150 IFC$="S",RO=A(RO,2)
1160 IFC$="E",RO=A(RO,3)
1170 IFC$="W",RO=A(RO,4)
1180 RETURN
1190 '
1200 'GET THINGS
1210 FL=0
1220 FORJ=1TO5
1230 IFP(J)<>0,FL=FL+1
1240 NEXTJ
1250 IFFL=5,PRINT"YOU HAVE 5 OBJECTS,THAT'S ENOUGH":RETURN
1260 'ANY THINGS HERE?
1270 IFA(RO,5)+A(RO,6)+A(RO,7)>0,1310
1280 PRINT"I CAN'T SEE ANYTHING TO GET":RETURN
1290 '
1300 ' GET OBJECT
1310 D$="":E$="":F$=""
```

```
1320 D$=LEFT$(O$(A(RO,5)),3)
1330 E$=LEFT$(O$(A(RO,6)),3)
1340 F$=LEFT$(O$(A(RO,7)),3)
1350 IFC$=D$,N=A(RO,5):A(RO,5)=0:GOTO1380
1360 IFC$=E$,N=A(RO,6):A(RO,6)=0:GOTO1380
1370 IFC$=F$,N=A(RO,7):A(RO,7)=0
1380 J=1
1390 IFP(J)=0,P(J)=N:GOTO1410
1400 IFJ<5,J=J+1:GOTO1390
1410 PRINT"  YOU NOW HAVE THE ";O$(N)
1420 RETURN
1430 '
1440 'DROP OBJECT
1450 FL=0
1460 FORJ=1TO5
1470 IFP(J)<>0,FL=1
1480 NEXTJ
1490 IFFL=0,PRINT"YOU'RE NOT CARRYING ANYTHING":RETURN
1500 IFA(RO,5)*A(RO,6)*A(RO,7)=0,1520
1510 PRINT"THIS ROOM IS FULL OF THINGS":RETURN
1520 D$="":D=0
1530 FORJ=1TO8
1540 IFLEFT$(O$(J),3)=C$,D$=O$(J):D=J
1550 NEXTJ
1560 IFD$<>"",1580
1570 PRINT"YOU CAN'T,YOU HAVEN'T GOT IT !":RETURN
1580 FORJ=1TO5
1590 IFP(J)=D,P(J)=0
1600 NEXTJ
1610 IFA(RO,5)=0,A(RO,5)=D:GOTO1640
1620 IFA(RO,6)=0,A(RO,6)=D:GOTO1640
1630 IFA(RO,7)=0,A(RO,7)=D
1640 PRINT"YOU'VE DROPPED THE ";D$
1650 RETURN
1660 '"
1670 'FIGHT
1680 'MONSTER?
1690 IFA(RO,8)<>0,1750
1700 R=RND(0)
1710 IFR<.5,PRINT"FIGHT WHAT---WHERE ?":RETURN
1720 PRINT"YOU CAN'T FIGHT EMPTY AIR":RETURN
1730 '"
1740 'GET MONSTER NAME,SET STRENGTHS
1750 G$=M$(A(RO,8))
1760 S1=INT(RND(0)*6+RND(0)*6)+3
1770 H1=INT(RND(0)*6+RND(0)*6)+3
1780 D1=INT(RND(0)*6+RND(0)*6)+3
1790 I1=INT(RND(0)*6+RND(0)*6)+3
1800 W1=INT(RND(0)*6+RND(0)*6)+3
1810 C1=INT(RND(0)*6+RND(0)*6)+3
1820 PRINT"================================
1830 PRINT"   YOU FIGHT A ";G$
1840 MT=0:HT=0
1850 FF=S1*INT(RND(0)*6)+1
1860 PRINT"================================
1870 PRINT"THE ";G$;"'S DANGER LEVEL=";FF
1880 PRINT"================================
1890 FORZ=1TO1250:NEXTZ
1900 '" WEAPONS?
1910 FORJ=1TO5
1920 T(J)=0
1930 IFP(J)=6,PRINT"YOUR AXE MAY BE HANDY":T(J)=6
```

```
1940 IFP(J)=5,PRINT"THAT SWORD LOOKS USEFULL":T(J)=5
1950 NEXTJ
1960 FL=0
1970 FORJ=1TO5
1980 IFT(J)<>0,FL=FL+1
1990 NEXTJ
2000 IFFL<>0,2020
2010 PRINT"YOU FIGHT THE ";G$;" WITH":PRINT"YOUR BARE HANDS":GOTO2170
2020 IFFL>1,2090
2030 FORJ=1TO5
2040 IFT(J)<>0,FL=T(J)
2050 NEXTJ
2060 PRINT"YOU NEED YOUR ";O$(FL)
2070 FF=INT(FF*2/FL)
2080 GOTO2170
2090 PRINT"CHOOSE YOUR WEAPON:-
2100 FORJ=1TO5
2110 IFP(J)<>0,PRINTJ;"-";O$(P(J)),
2120 NEXTJ:PRINTTAB(0);
2130 INPUT"WHICH NUMBER";J
2140 IFP(J)=0,PRINT"YOU HAVEN'T GOT IT ! !":SOUND4,2;2,4:GOTO2130
2150 PRINT"OKAY THEN,ITS THE ";O$(P(J))
2160 FF=INT((FF*2/J)
2170 FORZ=1TO1500:NEXTZ
2180 PRINT"==============================
2190 PRINT"THE ";G$;" HAS THESE":PRINT"ATTRIBUTES:-
2200 PRINT"▮,STRENGTH ";S1,"▮,CHARISMA ";H1
2210 PRINT"▮,DEXTERITY";D1,"▮,INTELLECT";I1
2220 PRINT"▮,WISDOM   ";W1,"▮,VIGOUR   ";C1
2230 PRINT:PRINT"  YOUR ATTRIBUTES ARE:-
2240 PRINT"▮,STRENGTH ";ST,"▮,CHARISMA ";CH
2250 PRINT"▮,DEXTERITY";DE,"▮,INTELLECT";IT
2260 PRINT"▮,WISDOM   ";WI,"▮,VIGOUR   ";CO
2270 PRINT:PRINT"WHICH TWO ATTRIBUTES WILL YOU   FIGHT WITH";
2280 INPUTZ,Q
2290 IFZ*Q>0ANDQ<7ANDZ<7ANDZ<>Q,2310
2300 SOUND27,1;25,1;27,1;25,1:PRINT"DON'T MESS ABOUT ! ! THERE'S A
2305 PRINTG$;" IN THERE !":GOTO2280
2310 IFZ=1ORQ=1,MT=MT+S1:HT=HT+ST
2320 IFZ=2ORQ=2,MT=MT+H1:HT=HT+CH
2330 IFZ=3ORQ=3,MT=MT+D1:HT=HT+DE
2340 IFZ=4ORQ=4,MT=MT+I1:HT=HT+IT
2350 IFZ=5ORQ=5,MT=MT+W1:HT=HT+WI
2360 IFZ=6ORQ=6,MT=MT+C1:HT=HT+CO
2370 IFHT=MT,PRINTTAB(3);"YOU ARE EVENLY MATCHED-":GOTO2420
2380 PRINT"   THE ODDS SEEM TO FAVOUR ";
2390 '
2400 IFHT>MT,PRINT"YOU"ELSEPRINT"THE",G$
2410 '
2420 PRINT:PRINT"------------------------------
2430 '"▮▮▮ FIGHT CONTROL
2435 KK=0
2440 K=INT(RND(0)*7):IFK=KK,SOUND19,1:GOTO2440
2445 KK=K
2450 FORZ=1TO900:NEXTZ
2460 PRINT"THE ";G$;" =";MT
2470 PRINT"            YOU  =";HT
2480 PRINT"------------------------------
2490 IFK=0,PRINT"YOU STRIKE A GREAT BLOW":MT=MT-2
2500 IFK=1,PRINT"THE ";G$;" STRIKES":HT=HT-1:CH=CH-1
2510 IFK=2,PRINT"YOU DRAW ";G$;"'S BLOOD":ST=ST-I:MT=MT-3
2520 IFK=3,PRINT"YOU ARE WOUNDED !":IT=IT-1:HT=HT-1
```

CONTINUED Next Issue.

STRUCTURED PROGRAMMING - Part 3 by Chris Hobrough.

In Part 1 of this series of articles we saw that the process of designing a program can be simplified by adopting a top-down, modular approach. The program is reduced to a small number of steps in a main controlling module, each step becoming the task of a major sub-module. These modules are then further sub-divided until we reach the level of individual lines of code.

Part 2 looked at the flow of logic within the individual modules and at some of the formal command structures that help contribute to clear, understandable code. These structures are used so frequently that they soon become second nature for the programmer and are easily recognized by anyone trying to decipher the program later on. The use of these techniques not only makes the program easier to design and debug, it actually reduces the number of errors created.

It may seem then, that the job is finished. After all, the primary aim when designing a program is to produce something that does what it was intended to do. In reality it's only half done. Almost as important is the way the program is documented. Programs for serious use often need to be maintained or modified by other programmers and even those written by hobbyists for personal use will sometimes be passed on or perhaps looked at some time later in order to re-use ideas. If the flow of logic in the program is to be understood it must be well documented.

Documentation

Documentation should be clear and able to fully explain what's happening. One of the major side benefits of writing with pseudo-code initially is that the bulk of the job is already done. As we saw when the main control module was translated to BASIC, the pseudo-code was simply moved over and placed in REM statements. If anything, some comments can be left out because the code is self explanatory. PRINT statement are a good example. In addition, because the pseudo-code is written as you go, the temptation to leave the documentation until last is avoided and, consequently, the tendency to be a too sketchy is overcome.

There are still one or two things to do, however, to make the program really understandable. Firstly, it's a good idea to give it a general header containing the name, date and a brief description of the program's function. Since only the first two letters of variable names are significant on the VZ, it's usually safer not to use full length names. This can be confusing when reading the program, so a key to variable names can be very helpful.

When you come to look back at a program for routines that can be re-used, it's very helpful to know what variables are passed to and from each routine. This can be done with notes at the start of each routine, where necessary, listing the input and output parameters.

Lastly, the overall presentation can be much improved by visually separating modules in some way and by spacing out blocks of code with REM statements. A simple row of dots is less cluttering than anything too fancy and you really can't beat plain space. Proper indentation of structures can be very important, especially where there are nested loops or complex IF ... THEN structures.

The main problem with this sort of presentation, for the VZ user, is that it can often double or treble the size of the program. The solution lies in the fact that programs should be written on paper, not at the keyboard. We all probably learned BASIC by typing routines directly into the computer and running them to see what would happen. That is one of the advantages of BASIC but it really is no way to write programs of more than a few lines. A program should be designed, written and tested on paper before going near the computer. Once done, then only the actual code need be entered. The fully documented version stays on paper. One word of warning though. Remember that a

GOSUB or GOTO to a REM line will crash if the line is removed, so adjust accordingly or plan ahead.

Conclusion

BASIC has never been very popular with professional programmers because of its lack of structure and speed. Added to that is the tendency, among BASIC programmers, to try to fit as many statements on one line as possible and to use GOTO in a totally indiscriminate way. The unreadable mess of code that results is often referred to as "spaghetti code" - for obvious reasons. However, it's precisely the lack of imposed structure which makes BASIC one of the most versatile languages you can use. The problem is that you have to provide the structure yourself and the techniques discussed in this series of articles can help you do that.

I would've liked to develop our data base program a lot further - both the pseudo-code and BASIC versions - but it would've involved a lot of repetition and more space than is really warranted. The extracts I've chosen illustrate the basic ideas and I hope that I've succeeded in demonstrating their value. By following the general principles outlined above, programming becomes a matter of applying practiced technique rather than hit and miss cleverness. The way these principles are used will depend on the nature of the program and on individual style but it is possible to give some general guide lines.

When planing the program start at the most general level and identify the significant steps. This gives you your main module and first level sub-modules. Repeat the process by braking each step down into smaller steps until you're at the level of individual lines of code. If a module turns out to have only a couple of lines it can be brought back and incorporated into its parent module. Conversely, a module that is too large to manage can be split up.

Within the modules use clearly defined command structures. The GOTO statement is the single most abused command in BASIC but it is also the most powerful. Without branching commands (especially conditional ones) computers would never have advanced beyond the level of curiosity pieces and all of the formal command structures that modern languages provide have implied GOTO's in them. To avoid the problems that the undisciplined use of GOTO can cause, use it only to simulate the structures you don't have and never use a GOTO to a point outside the module. As with anything in programming, there is more than one way to simulate some of these structures, especially the IF structures. My illustrations were chosen to most clearly demonstrate the flow of logic and are not necessarily the most efficient way to solve the particular logic problem.

Finally, I find that a line numbering strategy helps me remember where things are. I usually start major routines on even thousands and minor ones on even hundreds. 0 - 99 is for the header and 100 - 999 is for such things as moving Top of Memory and other system housekeeping stuff. The arrangement doesn't matter as long as you develop a system you like and stick to it.

This approach to program design can only be fully appreciated by trying it. With a little practice it becomes second nature and you wonder how you managed to keep things organized without it. Even if it only influences your initial thoughts about a programming project it can make quite a difference. I hope you will find it of value.

DISK DRIVE PROBLEMS . . . . . .

One of our club members had an unusual problem with his drive. It didn't matter what disk command was used the drive motor would go on and on and not turn off till power was turned off. Also no disk function was performed. The only function that worked correctly was the error message to inform you DISK WRITE PROTECTED.

Investigation by club member revealed a 5 cent coin inside drive. It seems his child used his drive as a money box, a very expensive one too. I offered to have a look at drive as I had some ideas on possible problem.

The VZ disk drive system consists of Drive and Controller and obviously both had to be tested. First, his disk controller was tried with my drive and it worked OK. Next my controller was tried with his drive and it still had faults as before which ruled out his controller.

Fixing anything is a process of elimination and deduction and sometimes a lot of luck. By studying circuit diagram of drive in Tech. Ref. Manual I suspected U8, ULN2003 which is an octal relay/motor driver IC. The other contender was a 74LS14, a HEX SCHMITT TRIGGER IC. While the circuit board was out I decided to remove four IC'S and put in sockets in case others had to be replaced.

Surprise !, a 74LS32 was discovered on P.C.B. which wasn't shown in circuit and I had no idea of function it performed. The 74LS14 was replaced first and it made no difference. Next the 74LS32 and ULN2003 were replaced and drive worked OK. As it was getting late, drive was reasembled and I haven't had chance to find out which IC was the culprit. I strongly suspect the ULN2003.

It appears that early versions of disk drives have no 74LS32'S while later ones have and both my drives do.

DOS PROBLEMS . . . . . .

As editor I receive and send disks all over Aust. and N.Z. Mostly the disks work OK, but a few do not. I used to blame drive incompatibility, but couple days ago I came across second version 1.2 DOS which previously I only heard about. As far as I know there are five versions of DOS around and I suspect not all 100% compatible with each other. The five versions are -

V1.0 - V1.1 - V1.2A - V1.2B - Laserlink Dos.

I have version 1.0 and 1.2B. I call the two versions of V1.2, A and B simply to tell them apart as they both power up as V1.2. One of the programs sent to me used to hang up the VZ when specific disk access was performed. I placed V1.2A in my 8K BIB Ram and switched out V1.2B and switched in V1.2A, and now the particular program performs as designed.

When time permits I'll investigate as far as I can the incompatibility between the two versions, or if someone already has, then we would be pleased if you would share with other VZ users. I've never came across a V1.1 DOS and if someone has one, then I would like to hear from you.

PROGRAMMING

THE % ODDS IN LOTTO

- by Ken CLARKE.

[As background to this article, refer to K. Wright's "Lotto Program" in the September, 1987 Issue No. 12.]

If you haven't got around to typing in this program that does a grand job of printing-u the latest predictions, then maybe you might be looking for a copy already typed for yo I would be pleased to arrange free disc copies either at the next Work-Shop or you coul

```
1 LPRINTCHR$(27);CHR$(83);CHR$(1);:LPRINTCHR$(27);CHR$(15);
2 LPRINTCHR$(27)+"1":PAGE(113):FEED(14):LTAB(10)
10 G=0:F=5:E=5:D=4:C=3:B=2:A=1
20 F=F+1
30 IFF>40THENF=E+1:E=E+1
40 IFE>39THENE=D+1:D=D+1
50 IFD>38THEND=C+1:C=C+1
60 IFC>37THENC=B+1:B=B+1
70 IFB>36THENB=A+1:A=A+1
75 IFA>35THENLPRINT"   ***   T H E   E N D   ***":STOP
80 IFA=BORA=CORA=DORA=EORA=FTHENGOTO20
90 IFB=AORB=CORB=DORB=EORB=FTHENGOTO20
100 IFC=AORC=BORC=DORC=EORC=FTHENGOTO20
110 IFD=AORD=BORD=CORD=EORD=FTHENGOTO20
120 IFE=AORE=BORE=CORE=DORE=FTHENGOTO20
130 IFF=AORF=BORF=CORF=DORF=ETHENGOTO20
160 G=G+1
170 LPRINTUSING" ## ";A;B;C;D;E;F;:LPRINT,
175 PRINTUSING" ## ";A;B;C;D;E;F;:PRINT,
180 IFF=40THENLPRINT"* TOTAL = ";G:LPRINT
185 IFF=40THENPRINT"* TOTAL = ";G:PRINT
190 GOTO20
```

ask Peter. I have made a number of enhancements to the original program to improve it's presentation.

And now to the program above. Have you ever asked yourself: "What are my chances of picking the winning combination?" The answer to this you will find in my program. Try it and see for yourself just what the percentage odds really are. The program will keep it's own tally as it progresses. (Note that 'Printer Patch' was used to help page the printing. To run the program without this, you will need to delete the last three statements in line 2.)

ACKNOWLEDGEMENTS

Monster Hunt written by Keith Wright.
Disk and Dos Problems by Joe Leon.
Games Review
RePrinted with permission of hunter valley vz users club.

VZ-200 BASIC PROGRAM STORAGE & LINE RENUMBERING

GRAHAM MARSDEN

The VZ-200 does not have a RENUMBER command so trying to modify a program with insufficient vacant line numbers is not a welcome task. This program enables the line numbers of a program to be reset using any start number and increment providing they meet certain conditons.

In order to understand the operation of the program it is necessary to understand how a BASIC program listing and its line numbers are stored in memory.

Each line of program is formated as below:-

- The first two bytes of the sequence hold the address, in two byte form, of the first byte of the sequence for the next program line. i.e. the location holding the R above is in location P+256*Q
- The third and fourth bytes hold the line number.
  i.e. in this case the line number will be L+256*I
- Then the contents of the program line follow, terminating with a byte containing the value zero.

For example suppose the line:-
300 PRINT"!":GOTO400
was stored starting at address 38420 - 38433

Note that characters (including line numbers used within a program line after GOTO or GOSUB) are stored as their ASC codes.

"Operators" like PRINT, GOTO, etc have their own single byte codes which represent the operation. The program looks for the codes for GOTO and GOSUB (amongst others- see explanation of program operation) in order to find the locations of line numbers within program lines. The codes for various operations can be determined by putting in a line. using the operation in question, (ensure it has the lowest of all line numbers) and then type in -

FORZ = 31469 TO 31469+N:PRINT PEEK(Z);:NEXTZ
where N is the number of memory positions that you wish to see codes for. 31469 is the position in memory of the first item of the first BASIC program line. i.e. the one immediately after the line number bytes. The BASIC Program listing normally starts at 31465 unless moved - but that is another story.

Having understood how a BASIC program is stored it is possible to make changes to it without having to edit it on screen.

One thing that can be done is to change all the line numbers so they follow a constant incremant.

Here is a program to do just that:-

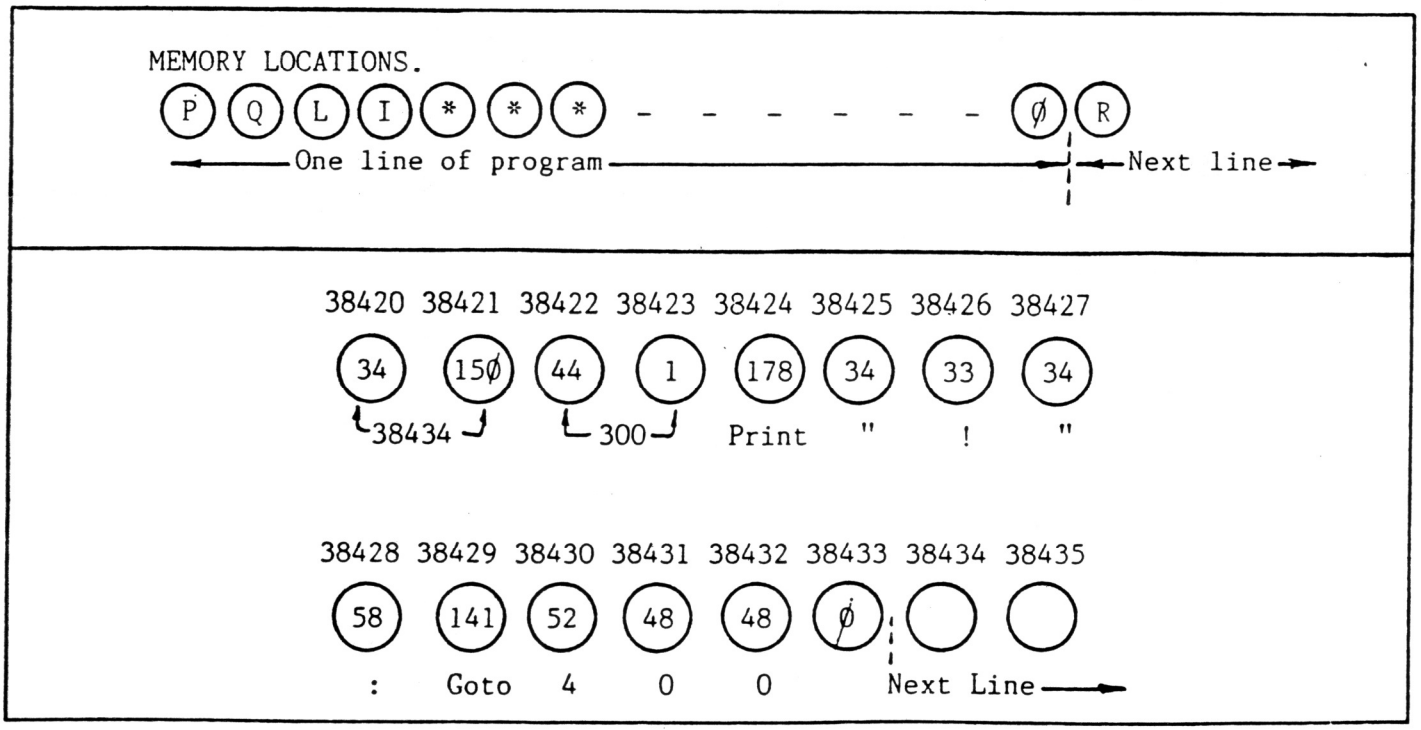
How to use this program:-

1) Type in and CSAVE as listed.
2) Before keying in your next program load the renumbering program from tape.

3) Key in your program with particular attention to the following.
   a) Line numbers used and called must be in the range 1-9999
   b) All line numbers or subroutines quoted within the text of a program line must be preceeded by GOTO or GOSUB and be right justified in a 4 space field. This means that 5 digit numbers if used will be seen only as the first four digits from the left and therefore will not be found as an existing number.

   i.e. IF ...THEN20ELSE325 must be entered as
   IF...THENGOTO 20ELSEGOTO 325
   (the line number 20 is preceeded by two spaces the number 325 by one, to create a 4 space field for the number - This allows say a two digit number to be reassigned as a three or four digit number)

   4 space field for the number - This allows say a two digit number to be reassigned as a three or four digit number)
   c) Line 10010:-
   Dimmension N%() greater than the number of line numbers in the section of program to be renumbered - A generous guess will do unless you are short of memory.
   :- Set the value of variable S to create a "safe zone" which the renumbering program will not alter. Normally this value will be 10000 (the first line number of the renumbering program itself) or it may be less if you wish to create a "gap" in line numbers between two sections of program - say between a main operating section and another section containing subroutines or Data lines. Remember that nothing in the "safe zone" is altered to a GOTO or GOSUB calling a lower section renumbered line would have to be changed separately.
4) Always CSAVE BEFORE running this program. If for any reason the renumbering is not totally successful then what remains of your program will probably be useless as part will be renumbered and part will not - equivalent to a population explosion of bags.
5) Key in RUN10000
   (If the result is a BAD SUBSCRIPT ERROR IN 10090 then increase the size of N%() in line 10010 - reloading will not be necessary as nothing has been altered yet.)
6) Enter 1st line number and increment on prompt.
7) When the renumbering is complete the cursor will return and the computer will be in READY mode (The time to execute



will be about 1½ seconds per program line)

8) Make any changes as indicated by messages printed during execution. (You can BREAK the execution to copy notes from the screen if it gets too full and then enter CONT to continue).
9) Probably a good idea to CSAVE again

10) Thoroughly test the renumbered section of the program - any problems - reload that saved at (4) and reRUN 10000 - it is not unknown for gremlins to be about for one renumbering run but absent for the next.

Program Operation

LINES 10000-10090

The line numbers are stored in array N%. Variable M is initialised to 31465 (BASIC Program start) and moved to the 1st byte of each program line by the calculation based on the values of P and Q the "start - position of next line" pointers. While M holds the decimal value of the memory positions the value of variable A is used in the PEEK'S. This is because PEEK and POKE will only work for the range -32768 A 32767 values for memory above 32767 must have 65536 subtracted before PEEK or POKE.
Line 10070 ends execution if P and Q both are zero - this occurs when the end-of-program byte sequence is found. (Two zero bytes then a 4 byte.

LINES 10100-10120

The first line number and increment are entered and edited so that they are both positive integers greater than zero. At 10120 a check is made as to whether the new numbering reach into the "safe zone" of line numbers.

LINE 10130

Reinitialise M back to program start and variable c to 1, C is the number of the line ie 1st 2nd 3rd etc.

LINES 10140-10180

Calculate new line number and POKE new line number to the appropriate bytes. M is now at the first byte of the line's storage sequence. The execution ends at 10160 if the "safe zone" is found.

LINES 10190-10210

Calculate R the memory position of the start of the next line.

LINES 10220-10290

This section searches through the program line contents looking at the value in each byte:

0: - end of line - go to next line
34: - Quote marks in a PRINT or INPUT statement - ignore
136: - DATA - ignore whole line
147: - REM - ignore whole line
141: - GOTO or 145 GOSUB: - alter line number called.

LINES 10300-10380

M set to the units column position of the number field area. The string variable 0$ is loaded with the characters of the line number field, and variable 0 is given the value of the

```
10000 '"VZ-200  LINE RENUMBERING G.A MAR
SDEN 1984
10010 DIM N%(200):M=31465:C=1:S=10000'"F
IRST LINE OF SAFE ZONE"
10020 A=M:IFA>32767 THEN A=A-65536
10030 P=PEEK(A):A=M+1:IF A>32767 THEN A=
A-65536
10040 Q=PEEK(A):A=M+2:IFA>32767 THEN A=A
-65536
10050 L=PEEK(A):A=M+3:IF A>32767 THEN A=
A-65536
10060 I=PEEK(A)
10070 IF P=0ANDQ=0THEN PRINT "SAFE LINE
PAST END  ":END
10080 IFL+256*I>=STHEND=C-1:PRINTD;"LINE
S FOUND":GOTO10100
10090 N%(C)=L+256*I:M=P+256*Q:C=C+1:GOTO
10020
10100 INPUT"1ST LINE N0.";L$:F=INT(VAL(L
$)):IF F<1 THENGOTO10100
10110 INPUT"INCREMENT";I$:J=INT(VAL(I$))
:IFJ<1THENJ=1
10120 IF F+J*(C-1)>=STHEN PRINT"VALUES T
OO LARGE":GOTO10100
10130 M=31465:C=1
10140 A=M+2:IFA>32767THENA=A-65536
10150 B=M+3:IF B>32767A=A-65536
10160 IF PEEK(A)+256*PEEK(B)>=STHEN PRIN
T"FINISHED":END
10170 W=F+(C-1)*J
10180 POKEA,W-256*INT(W/256):POKEB,INT(W
/256)
10190 A=M:IFA>32767 THENA=A-65536
10200 B=M+1:IFB>32767 THENB=B-65536
10210 R=PEEK(A)+256*PEEK(B)
10220 M=M+4:T=1
10230 A=M:IF A>32767 THEN A=A-65536
10240 IF PEEK(A)=136OR PEEK(A)=147 THEN
M=R:C=C+1:GOTO10140
10250 IFPEEK(A)=145OR PEEK(A)=141 THEN G
OTO10300
10260 IF PEEK(A)=34THEN T=T*-1
10270 IF PEEK(A)=0 THEN M=R:C=C+1:GOTO10
140
10280 M=M+1
10290 IFT<0 THENGOTO10260ELSE GOTO10230
10300 M=M+4
10310 0$=""
10320 FORG=3TO0STEP-1
10330 A=M-G:IFA>32767THENA=A-65536
10340 0$=0$+CHR$(PEEK(A))
10350 NEXTG
```

line number. Lastly a check to see if the line number called is within the safe zone.

LINES 10390-10470

The position of the old line no 0 is found in the arrayN%() H% is the position in the array that the value of 0 is compared with, in line 10470, and is initialised at the middle of the occupied area of the array. K% is initialised at just over¼ of the occupied length of the array. K% is reduced to just over ½ its value at each loop and H% is altered by adding or subtracing K% depending on which direction the search for the line number must go. The values of the last two array positions looked at are held in HL% and HP%. If a second look is taken at any array position then the conclusion is that the number does not exist and the error message at 10460 is printed and the search for another GOTO or GOSUB resumes at 10230. This search routine takes only 5 or 6 loops to find a number in an array of 70 line numbers and is therefore more efficient than just starting at the bottom and looking at each array position on the way up, which would take an average 35 loops to find a line number Of course it relies on the fact that the numbers are stored in numerical order. The tests at 10430 and 10440 are to see if the search has gone beyond the occupied range of the array and modify H% and K% accordingly. This routine is useful to look through any array of values providing they are in number order. (ascending or descending).

LINES 10480-10510

String variable NN$ is set to the characters of the new line number (including spaces) to be inserted in the 4 byte field of the program line. Line 10500 allows the start positions of subroutines to be recorded as the renumbering goes on. This line could be ommited and the start of subroutines marked in the program listing using REM" or ". The " allows the remarks to be put in inverse characters to make them stand out as the program zooms up the screen after LIST.

LINES 10520-10620

This segment ensures that the value of 0 is consistant with the number of spaces at the left of the 4 byte field that the number came from.

LINES 10630-10670

Character by character POKEing of the new line number to its position in the line format.

```
10360 O=VAL(O$)
10370 IFO>=STHEN PRINT"LINE";W;"(NEW):-"
;OELSE GOTO 10430
10380 PRINT"[TWELVE SPACES]---INSIDE SAF
E ZONE:M=M+1:GOTO10230
10390 HPx=0:HLx=0
10400 Hx=1+D/2:Kx=D/2
10410 Kx(Kx+1)/2:HPx=HLx:HLx=Hx
10420 Hx=Hx+SGN(0-Nx(Hx))*Kx
10430 IFNx(Hx)=OTHENHx=Hx-Kx:Kx=1
10440 IF Hx<1THENHx=1:Kx=1
10450 IFHx=HPxTHEN GOTO10460ELSEGOTO1047
0
10460 PRINT"LINE";W;"(NEW):- ";O;"NOT FO
UND":M=M+1:GOTO10230
10470 IFNx(Hx)<>OTHENGOTO10410
10480 NN$="[3 SPACES]"+STR$(F+(Hx-1)*J)
10490 A=M-4:IFA>32767 THENA=A-65536
10500 IF PEEK(A)=145THEN PRINT"NEW SUBR@
 ";NN$"CALLED@";W
10510 NN$=RIGHT$(NN$,4)
10520 IFO>=1000THENGOTO10630
10530 A=M-3:IFA>32767THENA=A-65536
10540 IF PEEK(A)<>32THENGOTO10610
10550 IF O>=100THENGOTO10630
10560 A=M-2:IFA>32767THENA=A-65536
10570 IFPEEK(A)<>32THENGOTO10610
10580 IFO>=10THENGOTO10630
10590 A=M-1:IFA>32767THENA=A-65536
10600 IF PEEK(A)=32THENGOTO10630
10610 PRINT"LINE";W;"(NEW):FIELD ERROR";
O
10620 PRINT"[4 SPACES]---CHANGE TO NEW N
O.: ";NN$:GOTO10670
10630 FORG=1TO4
10640 A=M-4+G:IFA>32767THENA=A-65536
10650 POKEA,ASC(MID$(NN$,G,1))
10660 NEXTG
10670 M=M+1:GOTO10230
```

P.O.BOX 1972 C.P.O AUCKLAND N.Z

(C) AUCKLAND VZ 300/200 COMPUTER USER'S CLUB 1986

ALL MATERIAL IN THIS NEWSLETTER IS SUBJECT TO COPYRIGHT..
CONTRIBUTED PROGRAMS & ARTICLES CAN BE REPRODUCED ON THE UNDERSTANDING THAT THEY ARE FOR THE PRIVATE USE OF SUBSCRIBERS ONLY.
COPYRIGHT IS ALWAY'S RETAINED BY THE AUTHOR.
THIS CLUB WILL NOT BE HELD RESPONSIBLE FOR THE BREECHING OF COPYRIGHT CAUSED BY THE PUBLICATION OF ANY MATERIAL SENT IN UNDER A FALSE NAME.

YOU ARE WELCOME TO PHONE ME IF YOU HAVE PROBLEMS